Version AT/6

EGT3
ENGINEERING TRIPOS PART IIB

Tuesday 11 May 2026 14.00 to 15.40

Module 4M26

ALGORITHMS AND DATA STRUCTURES

*Answer not more than **three** questions.*

All questions carry the same number of marks.

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

*Write your candidate number **not** your name on the cover sheet.*

STATIONERY REQUIREMENTS
Single-sided script paper

SPECIAL REQUIREMENTS TO BE SUPPLIED FOR THIS EXAM
CUED approved calculator allowed
Engineering Data Book

10 minutes reading time is allowed for this paper at the start of the exam.

You may not start to read the questions printed on the subsequent pages of this question paper until instructed to do so.

You may not remove any stationery from the Examination Room.

1 (a) (i) Provide the mathematical definition of Ω-notation (Big-Omega). [5%]

(ii) Why is the statement "the running time of algorithm A is at least $O(n^2)$" meaningless? How can we formally write that the lower bound of running time is proportional to n^2? [10%]

(b) The following Python code is a partially completed implementation of the `merge(A, p, q, r)` procedure in the Mergesort algorithm:

```
def merge(A, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    L = [0] * (n1 + 1)
    R = [0] * (n2 + 1)

    for i in range(n1):
        L[i] = A[p + i]
    for j in range(n2):
        R[j] = A[q + 1 + j]

    L[n1] = float('inf')
    R[n2] = float('inf')

    # Complete the merging logic below:
    i = 0
    j = 0
    for k in range(p, r + 1):
```

(i) Complete the implementation in Python. Your solution should correctly handle the sentinel values and ensure the merged result is placed back into the original list A at the correct indices. [35%]

(ii) Consider the main loop (using index k) in your `merge` implementation. State a suitable loop invariant for this loop and briefly explain how the maintenance property is satisfied. [15%]

(c) The worst-case running time $T(n)$ of the full Merge Sort algorithm is described by the recurrence:

$$T(n) = 2T(n/2) + \Theta(n)$$

(i) Use the Master Theorem to solve this recurrence for $T(n)$. [20%]

(ii) Sketch a recursion tree for an input size of $n = 8$. Label the cost at each level and show how they sum to the total complexity. [15%]

2 (a) (i) What is the average runtime for lookup with a hash table that uses chaining, under the assumption of simple uniform hashing? What is the worst-case lookup runtime with a hash table? Which conditions lead to the worst-case runtime? [10%]

(ii) Define the load factor α of a hash table. For a hash table using chaining, state its storage costs. [10%]

(iii) Consider a hash table of size $m = 7$ using chaining. The hash function is $h(k) = k \bmod 7$. Insert the keys $19, 26, 13, 48, 17, 6$ in the given order and show the final contents of the table. [20%]

(b) (i) Define the Greedy Choice Property and state why it is necessary for a greedy algorithm. [10%]

(ii) You are given a set of seven characters with the following frequencies:

Character	Frequency
A	8
B	24
C	23
D	13
E	18
F	25
G	1

Using the Huffman algorithm, show the sequence of merge steps and draw the final prefix-code tree. [30%]

(iii) Give a valid binary codeword assignment for the characters based on your tree. [15%]

(iv) Explain briefly why Huffman coding is classified as a greedy algorithm. [5%]

3 (a) Consider a Matrix class with the following listing:

```
from typing import Tuple
class Matrix:
    def __init__(self, rows: int, columns: int):
        self.values = [0 for _ in range(rows * columns)]
        self.rows = rows
        self.columns = columns

    def __getitem__(self, key: Tuple[int, int]) -> float:
        row, column = key
        return self.values[row * self.columns + column]

    def __setitem__(self, key: Tuple[int, int], value: float):
        row, column = key
        self.values[row * self.columns + column] = value

    def swap_rows(self, i: int, j: int):
        # incomplete

    def add_rows(self, i: int, j: int, scale: float, start=0):
        # incomplete
```

Write an implementation in Python for the `swap_rows` method, which swaps the positions of rows `i` and `j`. [25%]

(b) Write an implemention in Python for the `add_rows` method, which adds row `j`, scaled by `s`, to row `i`, starting at column `start`. [25%]

(c) Consider the Gaussian elimination algorithm (Algorithm 1) for matrices on the next page. Analyse the computational complexity of this algorithm and state the complexity using the Big O notation. You can assume that $n > m$. `AddRows` and `SwapRows` in this algorithm are the same as `add_rows` and `swap_rows` in parts (a) and (b). [30%]

(d) You are presented with four counters A, B, C, D, initially displaying the values $1, 5, 3, 2$, and four buttons. Pressing a button decreases the value of a fixed subset of the counters by 1, as follows:

$$\{B\}, \quad \{B, D\}, \quad \{C\}, \quad \{A, C\}.$$

Each button press has unit cost, and counters may not become negative. Our goal is to get all buttons to zero. One possible solution uses eight button presses: $\{B\}$ three times, $\{B, D\}$ two times, $\{C\}$ two times, and $\{A, C\}$ once.

(cont.

Your task is to describe an algorithm that finds the minimum number of button presses required to make all counters show zero. You may describe your algorithm in words or pseudocode. [20%]

Algorithm 1 Gaussian Elimination

```
1:  function GaussianElimination(A, m, n)
2:      h ← 1, k ← 1                          ▷ Initialisation of the pivot row and column
3:      while h ≤ m ∧ k ≤ n do
4:          p ← arg max_i Abs(A[i, k]) where i ≥ h          ▷ Find the k-th pivot
5:          if A[p, k] = 0 then
6:              k ← k + 1                          ▷ No pivot, increase pivot column
7:          else
8:              SwapRows(A, h, p)
9:              for i ∈ h + 1 ... m do
10:                 f ← -A[i, k] / A[h, k]
11:                 A[i, k] ← 0
12:                 AddRows(A, i, h, f, k+1)
13:             h ← h + 1                          ▷ Increase pivot row
14:             k ← k + 1                          ▷ Increase pivot column
15:     BackSubstitute(A, m, n)          ▷ Back substitute the row-echelon matrix
16: function BackSubstitute(A, m, n)
17:     p ← 1
18:     for c ∈ 1 ... n do
19:         if p > m then
20:             return
21:         if A[p, c] ≠ 0 then
22:             for r ∈ 1 ... p - 1 do
23:                 f ← -A[r, c] / A[p, c]
24:                 AddRows(A, r, p, f, c)
25:             p ← p + 1
```

4 Following severe flooding, an emergency evacuation plan is being developed for a region consisting of several cities connected by one-way highways. Every city contains residents who may need to evacuate, and some cities contain designated evacuation centres.

The information available to planners is (1) a list of cities, (2) a list of directed highways between cities, each with a capacity and travel time, and (3) a list of evacuation-centre cities. For each highway, capacity is defined as the maximum number of people per hour that can safely travel along it, and travel time as the average travel time along that highway.

(a) Explain how this evacuation planning problem can be modelled algorithmically. Your answer should clearly identify what abstraction should be used and how the key information should be represented. You do not need to describe any specific algorithm in this part. [15%]

(b) Emergency planners want to know whether every city has at least one evacuation route to an evacuation centre, ignoring capacities and travel times. Briefly describe an algorithm to answer this question and give a tight asymptotic bound on its runtime. [15%]

(c) For a given city C, planners want to compute the minimum travel time required to reach any evacuation centre.

(i) Complete the pseudocode in Algorithm 2 to compute this value using Dijkstra's algorithm. [30%]

Algorithm 2 Dijkstra's Algorithm

```
    INITIALISE-SINGLE-SOURCE(G, s)
1:  for each vertex v ∈ V[G] do
2:      d[v] ← ∞
3:      π[v] ← NIL
4:  d[s] ← 0

    RELAX(u, v, w)
    // Your implementation here

    DIJKSTRA(G, w, s)
5:  INITIALISE-SINGLE-SOURCE(G, s)
    // Your implementation here
6:  return d
```


(cont.

(ii) State its asymptotic runtime, and the key assumptions for Dijkstra's algorithm to be correct. [5%]

(d) Planners now ignore travel times and focus only on highway capacities. They want to compute the maximum total number of people per hour that can be evacuated from all cities to the evacuation centres combined.

(i) Which algorithm can be used to compute this value? Briefly explain how. [20%]

(ii) Give a tight asymptotic bound on its runtime. [5%]

(e) After computing the maximum evacuation rate, planners want to identify the most critical highways whose capacities bottleneck the total evacuation rate. Briefly explain how these highways can be identified from the result of part d) and state the asymptotic runtime of this identification step. [10%]

END OF PAPER

THIS PAGE IS BLANK